Set-3

1. Design a lexical Analyzer to find the number of whitespaces and newline characters.

Code:

```
%{
#include <stdio.h>
int whitespace_count = 0, newline_count = 0;
%}

%%

[ \t]+  { whitespace_count += yyleng; }
\n     { newline_count++; }
.      { } // Ignore other characters

%%

int main() {
  printf("Enter input (Ctrl+D to stop):\n");
  yylex();
  printf("Total Whitespaces: %d\n", whitespace_count);
  printf("Total Newlines: %d\n", newline_count);
  return 0;
}

int yywrap() {
  return 1;
}
```

2. Develop a lexical Analyzer to test whether a given identifier is valid or not.

Code:

```
%{
#include <stdio.h>
%}

%%

[a-zA-Z_][a-zA-Z0-9_]*   { printf("Valid Identifier: %s\n", yytext); }
[0-9][a-zA-Z0-9_]*       { printf("Invalid Identifier: %s\n", yytext); }
```

```
.                   { } // Ignore other characters

%%

int main() {
  printf("Enter an identifier: ");
  yylex();
  return 0;
}

int yywrap() {
  return 1;
}
```

3. Write a LEX program to identify the capital words from the given input.

Code:

```
%{
#include <stdio.h>
%}

%%

[A-Z]+   { printf("Capital Word: %s\n", yytext); }
[a-zA-Z0-9_]+  { } // Ignore non-capital words
.      { } // Ignore other characters

%%

int main() {
  printf("Enter input: ");
  yylex();
  return 0;
}

int yywrap() {
  return 1;
}
```

4. The lexical analyzer should ignore redundant spaces, tabs and new lines. It should also ignore comments. Although the syntax specification states that identifiers can be arbitrarily long, you may restrict the length to some reasonable value. Write a LEX

specification file to take input C program from a .c file and count t he number of characters, number of lines & number of words.

Input Source Program: (sample.c)

```
#include <stdio.h>
int main()
{
   int number1, number2, sum;
printf("Enter two integers: ");
scanf("%d %d", &number1, &number2);
sum = number1 + number2;
   printf("%d + %d = %d", number1, number2, sum);
 return 0;
}
```

Code:

```
%{
#include <stdio.h>
int char_count = 0, line_count = 0, word_count = 0;
%}

%%
// Ignore spaces, tabs, and new lines
[ \t]+   ;
\n     { line_count++; char_count++; }

// Ignore single-line comments
//.*     ;

// Ignore multi-line comments
/\*([^*]|\*+[^/*])*\*+/ ;

// Count words (identifiers, keywords, numbers, etc.)
[A-Za-z_][A-Za-z0-9_]* { word_count++; char_count += yyleng; }
[0-9]+ { word_count++; char_count += yyleng; }

// Count other characters
.     { char_count++; }

%%
int main(int argc, char *argv[]) {
  FILE *file;
```

```
    if (argc > 1) {
        file = fopen(argv[1], "r");
        if (!file) {
            printf("Cannot open file %s\n", argv[1]);
            return 1;
        }
        yyin = file;
    }
    yylex();
    printf("Characters: %d\n", char_count);
    printf("Lines: %d\n", line_count);
    printf("Words: %d\n", word_count);
    return 0;
}
```